

DZISIAJ LISTY, KROTKI I PRZYKŁAD ŚRODOWISKA  
GRAFICZNEGO

---

PYTHON 02

## DO CZEGO NAM TO W OGÓLE POTRZEBNE

Klasyczna zmienna w Pythonie przechowuje jedną wartość np. `x=10`.

Przy czym możemy sobie wyobrazić, że do przechowywania większej ilości danych będziemy potrzebowali jakiegoś zbioru danych. Czyli właśnie listy. We wszelkich poradnikach porównuje się to do listy zakupów. Do półki na towary itp.

Podstawowa forma listy:

```
nazwa_listy = []
```

WAŻNE SĄ TUTAJ NAWIASY KWADRATOWE - lista może być pusta. Można później dodawać elementy. Nie powinno się robić nazwy „list” - bo używa się tego słowa przy klasach później.

```
print(nazwa_listy) - wypisuje nam wszystkie elementy z listy
```

# ELEMENTY LISTY MOŻNA ZDEFINIOWAĆ OD RAZU:

```
nazwa_listy = ["jabłko", "banan", "śliwka"]
```

I oczywiście wydrukować:

```
print(nazwa_listy)
```



The image shows a code editor window with a file named `main.py`. The code inside the file is:

```
main.py
1 nazwa_listy= ["jabłko", "banan", "śliwka"]
2 print(nazwa_listy)
```

Below the code editor is a terminal window showing the output of running the script:

```
['jabłko', 'banan', 'śliwka']
```

## W LISTACH MOŻNA PRZECHOWYWAĆ RÓŻNE ELEMENTY

```
nazwa= [1, „a”, True, [], {a: „A”}]
```

Bezpiecznie, zwłaszcza na początku nie mieszać typów danych ale można.

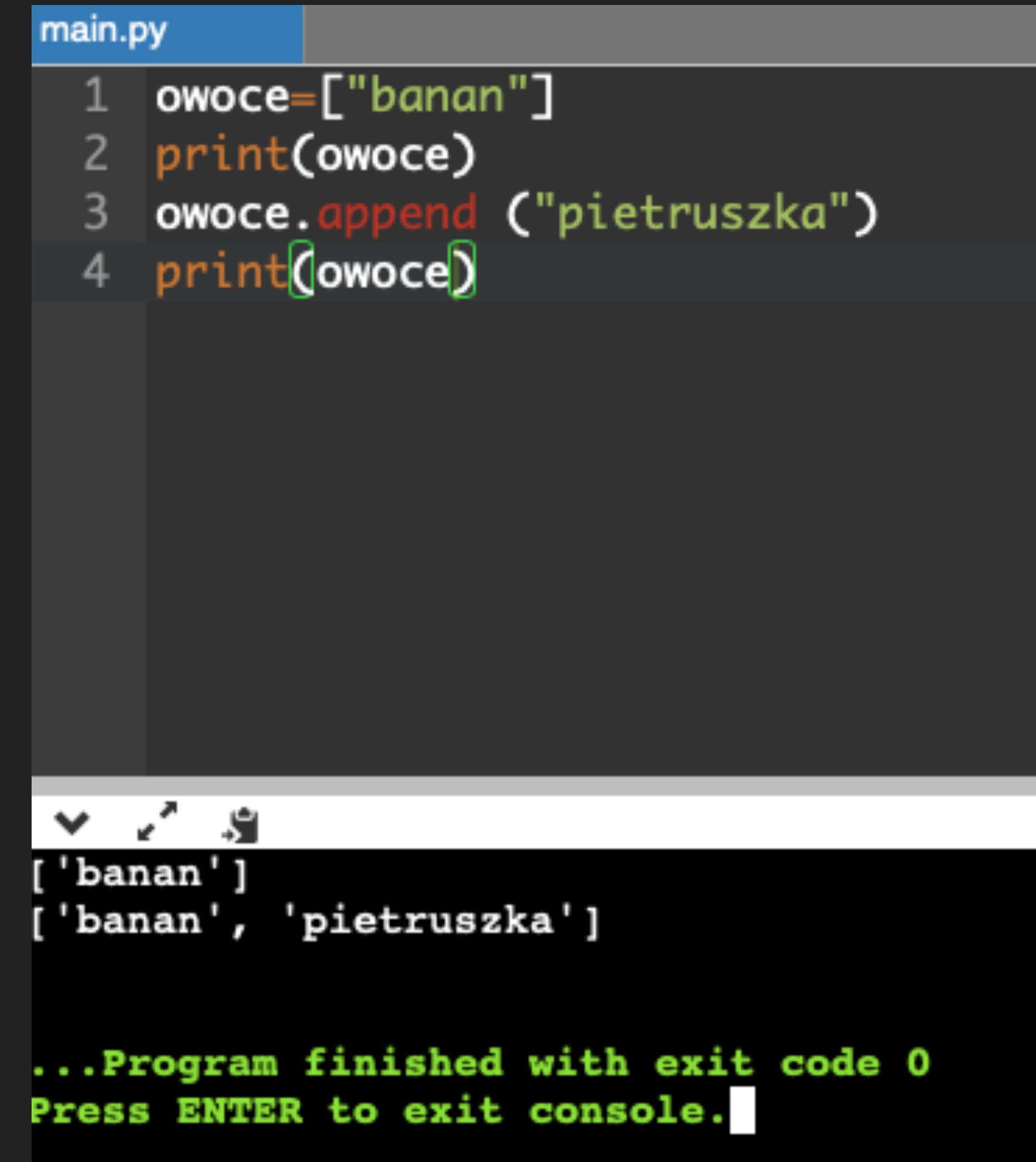
# DODAWANIE ELEMENTÓW DO LISTY

```
owoce = ["banan"]
```

```
print(owoce)
```

```
owoce.append("Śliwka")
```

Append dodaje na ostatniej pozycji



The image shows a code editor window titled "main.py" and a terminal window. The code editor contains the following Python script:

```
1 owoce=["banan"]
2 print(owoce)
3 owoce.append ("pietruszka")
4 print(owoce)
```

The terminal window shows the execution of the script. It first prints the list ["banan"], then adds the string "pietruszka" to the list using the append method, and finally prints the updated list ["banan", 'pietruszka']. The terminal concludes with a message indicating the program finished with exit code 0 and prompts the user to press Enter to exit the console.

```
['banan']
['banan', 'pietruszka']

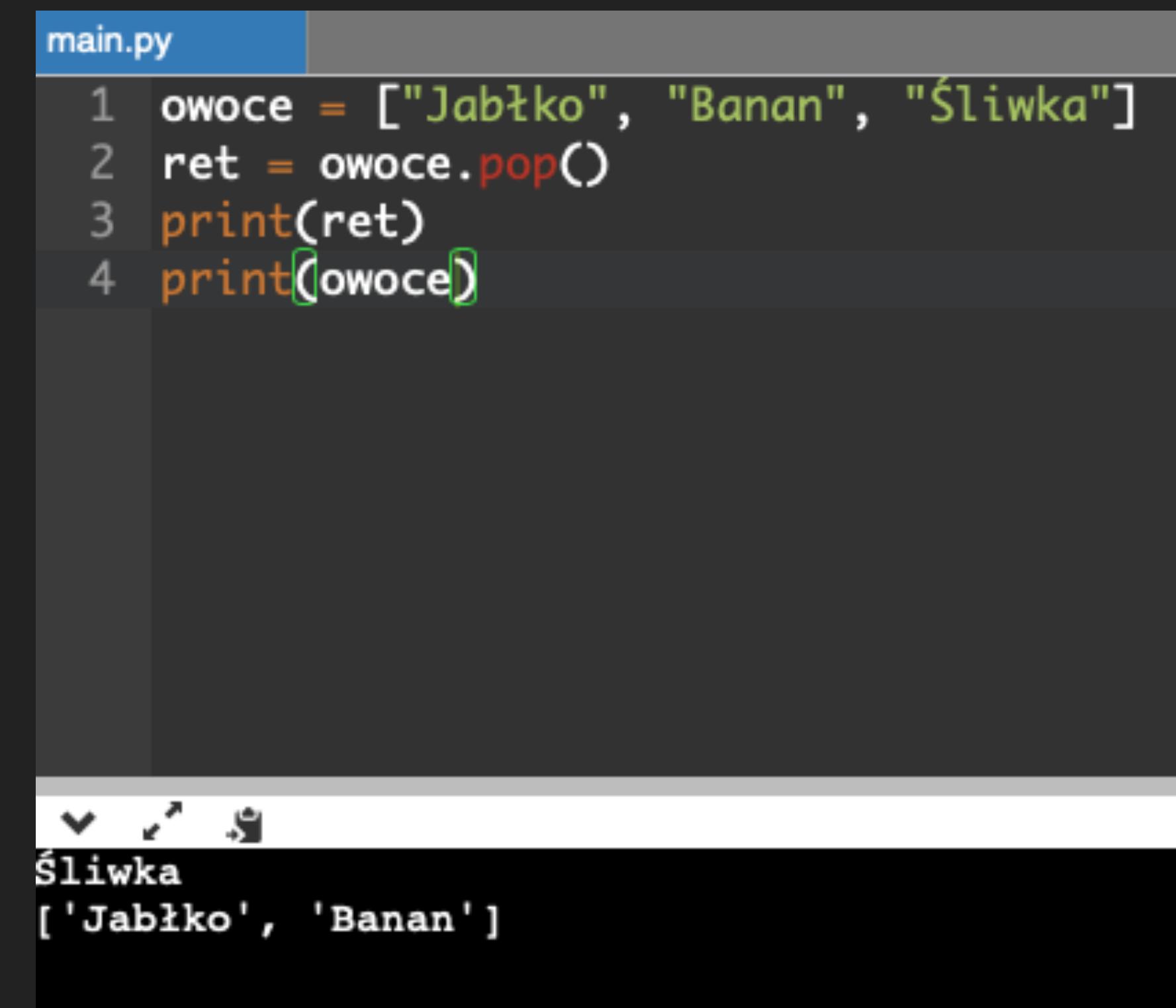
...Program finished with exit code 0
Press ENTER to exit console.
```

# WYCIĄGANIE ELEMENTÓW Z LISTY

```
owoce = [„Jabłko”, “Banan”, “Śliwka”]
```

```
ret = owoce.pop()
```

Ret wyciągnie nam owoc z listy, zarazem kasując.



```
main.py
1 owoce = ["Jabłko", "Banan", "Śliwka"]
2 ret = owoce.pop()
3 print(ret)
4 print(owoce)
```

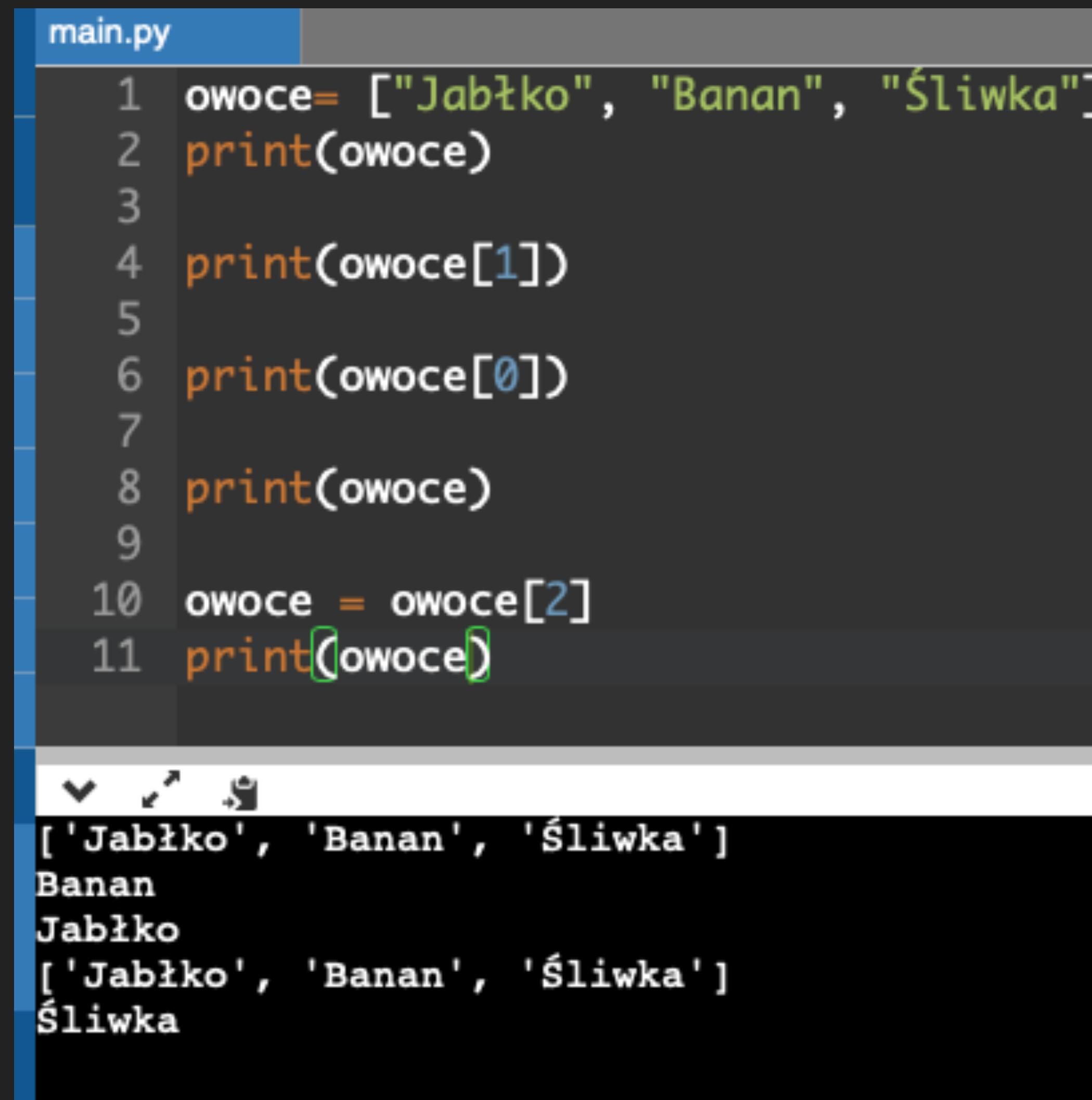
```
Śliwka
['Jabłko', 'Banan']
```

## POBIERANIE/ODNOSZENIE SIĘ DO ELEMENTÓW

Każdy element ma w liście przypisany numer. Nazywa się to indeksem. Numerem początkowym jest zawsze zero.

W Pythonie używa się terminologii, że element znajduje się na jakimś indeksie, to znaczy liczbie. I tak mamy listę [a,b,c,d]. Jak odniesiemy się do 0, to będzie a. Jak do 1, to b itd.

# POBIERANIE/ODNOSZENIE SIĘ DO ELEMENTÓW



The image shows a code editor window with a dark theme. The file is named 'main.py'. The code defines a list 'owoce' with three fruit names and uses print statements to output elements from the list. The output window below shows the list and the individual fruit names.

```
main.py
1 owoce= ["Jabłko", "Banan", "Śliwka"]
2 print(owoce)
3
4 print(owoce[1])
5
6 print(owoce[0])
7
8 print(owoce)
9
10 owoce = owoce[2]
11 print(owoce)

['Jabłko', 'Banan', 'Śliwka']
Banan
Jabłko
['Jabłko', 'Banan', 'Śliwka']
Śliwka
```

## POBIERANIE/ODNOSZENIE SIĘ DO ELEMENTÓW

Elementy mają swoje numery 0,1,2 itd.

Jeżeli chcemy wybrać ostatni element z listy, to ma on wartość „-1”, przedostatni „-2” itd. W ten sposób uproszczono tego typu operacje

# POBIERANIE/ODNOSZENIE SIĘ DO ELEMENTÓW

main.py

```
1 owoce = ["Jabłko", "Banan", "Śliwka"]
2 print(owoce)  # ["Jabłko", "Banan", "Śliwka"]
3
4 print(owoce[-1])  # Śliwka
5
6 print(owoce[-2])  # Banan
7
8 print(owoce)  # ["Jabłko", "Banan", "Śliwka"]
9
10 jedzenie = owoce[-3]
11 print(jedzenie)  # Jabłko
```

## POBIERANIE/ODNOSZENIE SIĘ DO ELEMENTÓW – ZAKRES

Elementy mają swoje numery 0,1,2 itd.

Zakres określamy poprzez podanie najpierw od jakiego indeksu elementy nas interesują, następnie dwukropka, a następnie przed jakim indeksem mamy skończyć. Na przykład, jeśli interesują nas elementy od drugiego do przedostatniego, to powinniśmy użyć indeksu drugiego elementu (czyli 1) oraz ostatniego (czyli -1), a więc 1:-1.

## POBIERANIE/ODNOSZENIE SIĘ DO ELEMENTÓW – ZAKRES

main.py

```
1 litery = ["A", "B", "C", "D"]
2
3 print(litery[1:-1])  # ['B', 'C']
4 print(litery[0:-2])  # ['A', 'B']
5 print(litery[0:-1])  # ['A', 'B', 'C']
```

## POBIERANIE/ODNOSZENIE SIĘ DO ELEMENTÓW – ZAKRES

Elementy mają swoje numery 0,1,2 itd.

Aby określić zakres, który nie ma górnego ograniczenia, nie stawiamy nic po prawej stronie dwukropka. Podobnie, jeśli po lewej stronie nie postawimy wartości, oznaczać to będzie zakres zaczynający się na samym początku. Samotny dwukropk oznacza więc zakres obejmujący wszystkie elementy.

## POBIERANIE/ODNOSZENIE SIĘ DO ELEMENTÓW - ZAKRES

main.py

```
1 litery = ["A", "B", "C", "D"]
2
3 print(litery[2:])  # ['C', 'D']
4 print(litery[1:])  # ['B', 'C', 'D']
5 print(litery[:2])  # ['A', 'B']
6 print(litery[:])  # ['A', 'B', 'C', 'D']
```

## ZMIANA ELEMENTÓW PRZY UŻYCIU INDEKSU

Elementy mają swoje numery 0,1,2 itd.

Zmiana elementu na indeksie wygląda identycznie jak odnoszenie się do niego, ale dodatkowo musimy postawić znak równości (oznaczający przypisanie) i nową wartość.

## ZMIANA ELEMENTÓW PRZY UŻYCIU INDEKSU

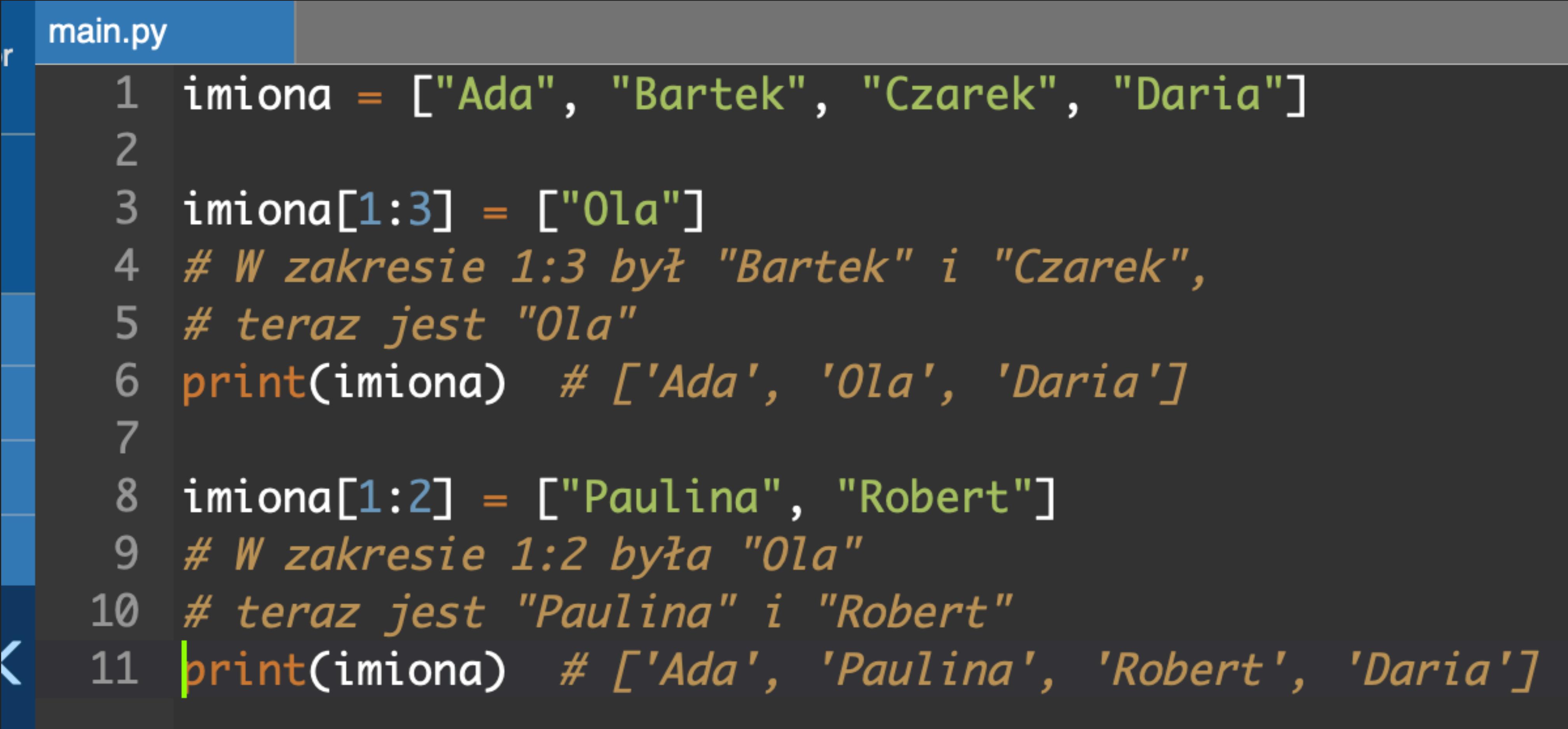
main.py

```
1 imiona = ["Ada", "Bartek", "Czarek"]
2 print(imiona) #oryginalna lista
3 imiona[1] = "Marek"
4 print(imiona) # ['Ada', 'Marek', 'Czarek']
5
6 imiona[-1] = "Nina"
7 print(imiona) # ['Ada', 'Marek', 'Nina']
```

## ZMIANA ELEMENTÓW PRZY UŻYCIU ZAKRESU

Analogicznie z zamianą elementów w zakresie. Tylko tutaj po prawej stronie powinniśmy postawić nową listę, której elementy powinny zastąpić elementy w podanym zakresie.

# ZMIANA ELEMENTÓW PRZY UŻYCIU ZAKRESU

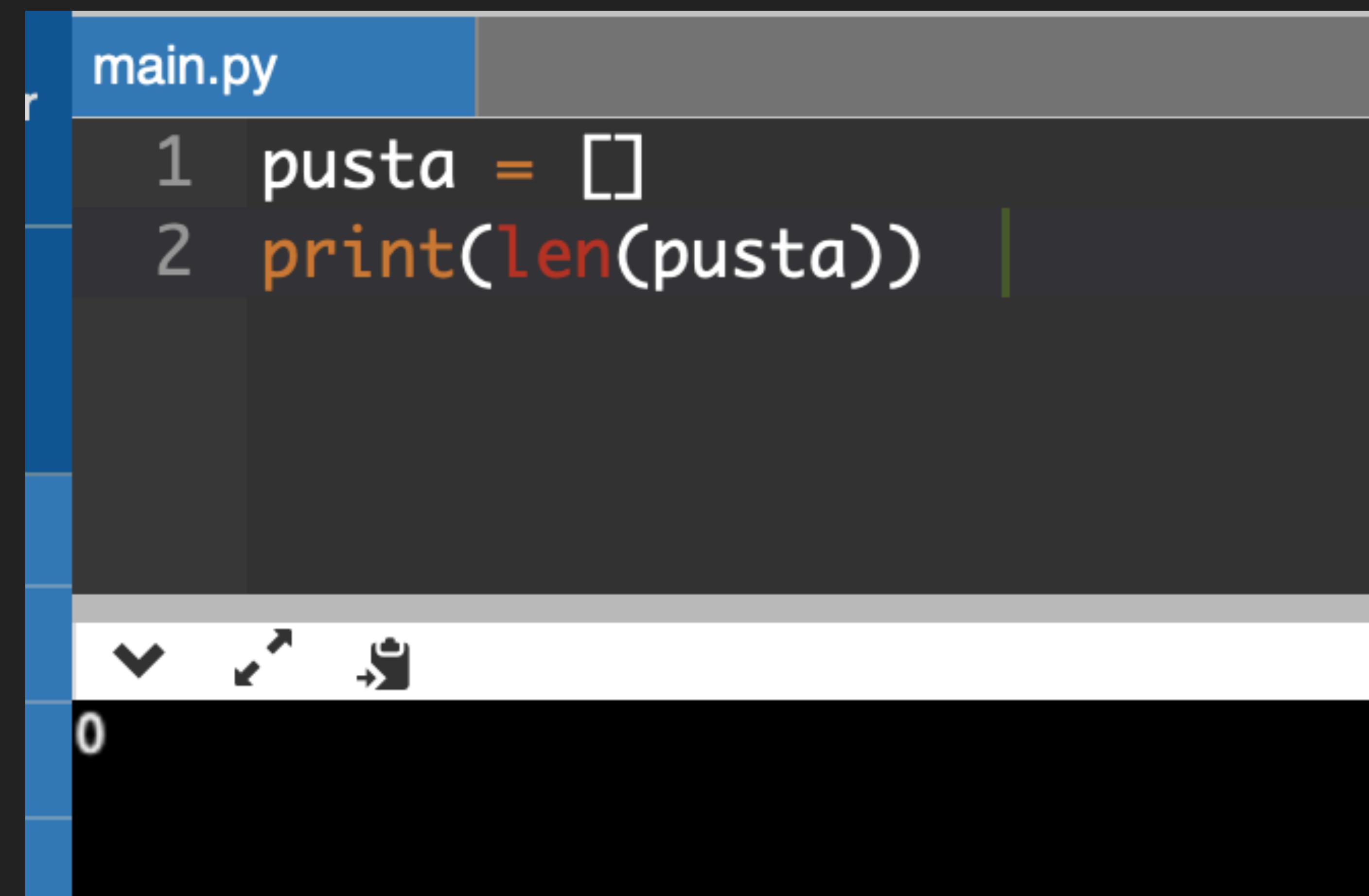


```
main.py
1 imiona = ["Ada", "Bartek", "Czarek", "Daria"]
2
3 imiona[1:3] = ["Ola"]
4 # W zakresie 1:3 był "Bartek" i "Czarek",
5 # teraz jest "Ola"
6 print(imiona) # ['Ada', 'Ola', 'Daria']
7
8 imiona[1:2] = ["Paulina", "Robert"]
9 # W zakresie 1:2 była "Ola"
10 # teraz jest "Paulina" i "Robert"
11 print(imiona) # ['Ada', 'Paulina', 'Robert', 'Daria']
```

## A JAK SPRAWDZIĆ ILE JEST ELEMENTÓW NA LIŚCIE (WIELKOŚĆ/DŁUGOŚĆ)

Do tego służy funkcja `len` (skrót z długość - podobnie jak w Excel):

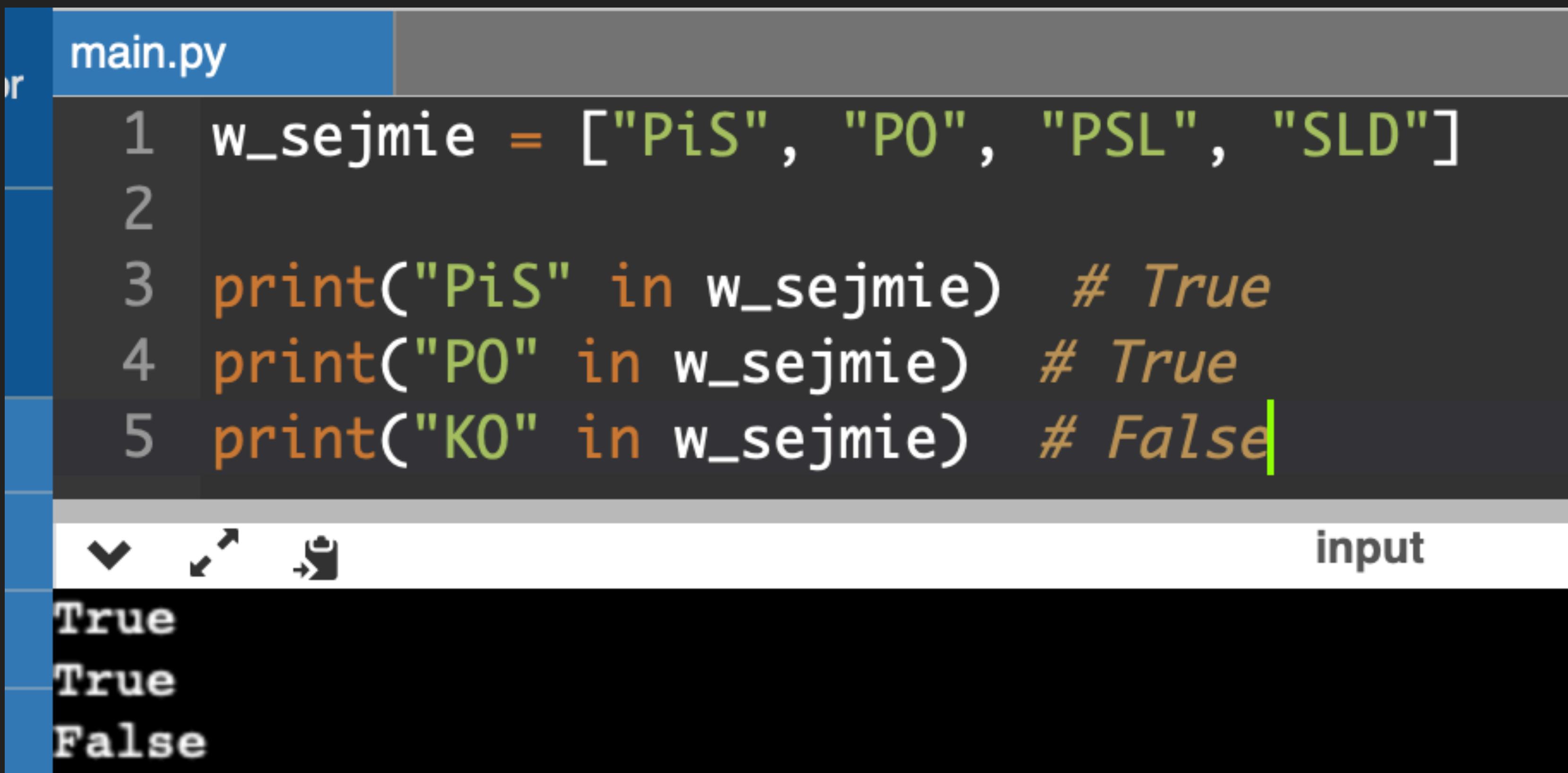
```
pusta = []
print(len(pusta))
```



```
main.py
1 pusta = []
2 print(len(pusta)) |
```

## A JAK SPRAWDZIĆ CZY JEST ELEMENT W/NA LIŚCIE

Służy temu element `in` pomiędzy elementem poszukiwanym oraz listą



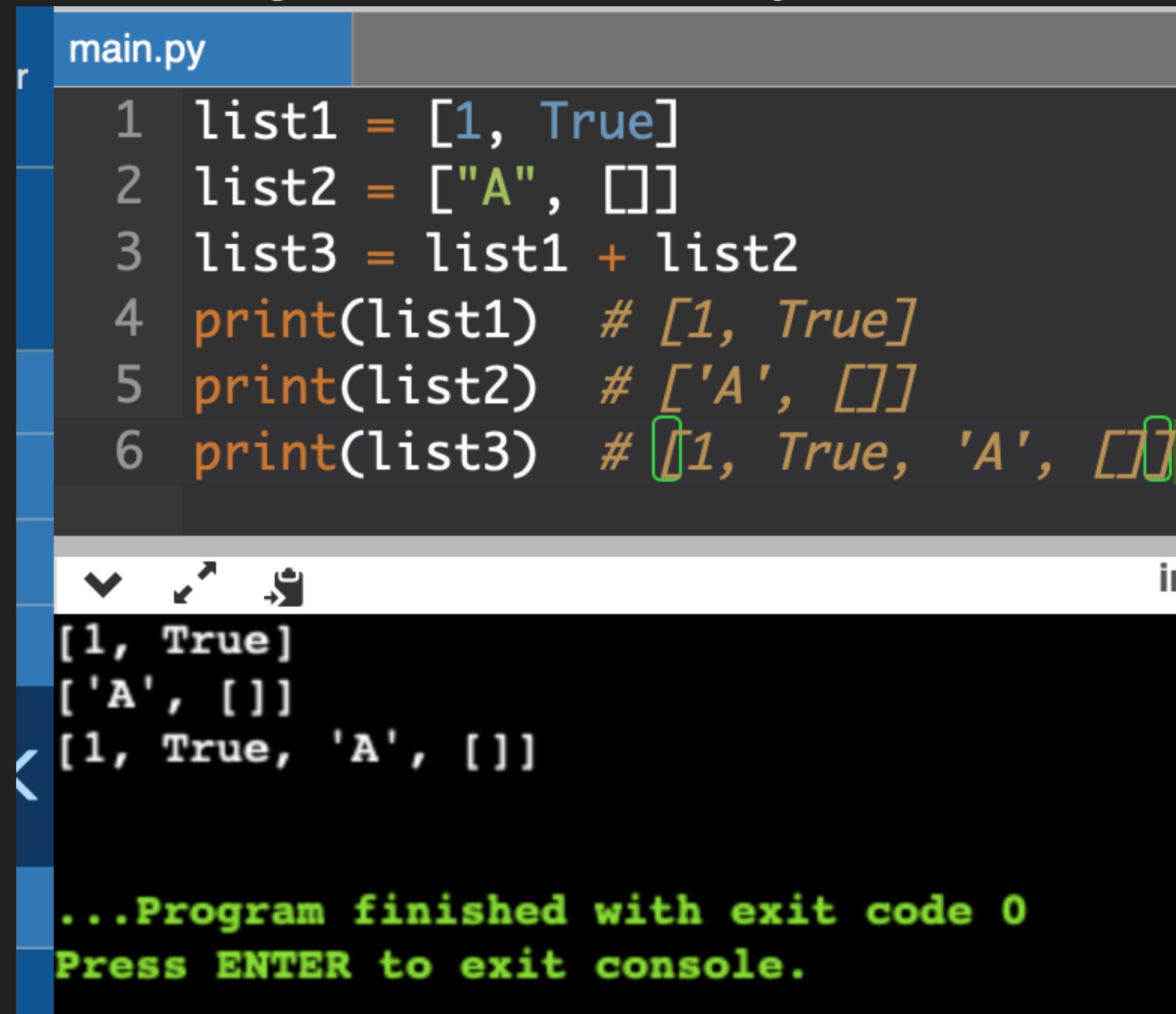
The image shows a code editor with a script named 'main.py' and its terminal output. The script defines a list 'w\_sejmie' containing political parties: 'PiS', 'PO', 'PSL', and 'SLD'. It then uses the 'print' function with the 'in' operator to check if 'PiS', 'PO', and 'K0' are in the list, resulting in 'True', 'True', and 'False' respectively. The terminal output shows the results of these three print statements.

```
main.py
1 w_sejmie = ["PiS", "PO", "PSL", "SLD"]
2
3 print("PiS" in w_sejmie) # True
4 print("PO" in w_sejmie) # True
5 print("K0" in w_sejmie) # False

input
True
True
False
```

## DODAWANIE I KOPIOWANIE...LIST

Skoro możemy dodawać stringi. Skoro możemy dodawać liczb, to czemu nie listy



The screenshot shows a code editor window with a dark theme. The file is named 'main.py'. The code contains six lines of Python code:

```
1 list1 = [1, True]
2 list2 = ["A", []]
3 list3 = list1 + list2
4 print(list1) # [1, True]
5 print(list2) # ['A', []]
6 print(list3) # [1, True, 'A', []]
```

Below the code editor, the terminal window shows the output of the program:

```
[1, True]
['A', []]
[1, True, 'A', []]
```

At the bottom of the terminal, the text "...Program finished with exit code 0" and "Press ENTER to exit console." is displayed.

# KROTKI

To niezmienna (immutable) struktura danych, która służy do przechowywania uporządkowanego zbioru elementów różnego rodzaju. Krotki są podobne do list, ale w odróżnieniu od list nie można ich modyfikować po ich utworzeniu. Elementy krotki są przechowywane w określonej kolejności, a do nich można się odwoływać za pomocą indeksów. Krotki są przydatne w sytuacjach, w których dane nie powinny ulec zmianie po ich zdefiniowaniu, na przykład jako klucze w słownikach lub elementy zestawów danych, które nie powinny być przypadkowo zmodyfikowane.

### KROTKI

main.py

```
1 moja_krotka = (1, 2, "trzy", 4.0)
2 print(moja_krotka[0]) # Wyświetla pierwszy element krotki (1)
3 print(moja_krotka[2]) # Wyświetla trzeci element krotki ("trzy")
4
5 # Krotki są niezmienne, próba zmiany elementu spowoduje błąd
6 # moja_krotka[1] = 10 # To spowoduje błąd TypeError
7
8
```

input

```
1
trzy
```

```
...Program finished with exit code 0
Press ENTER to exit console.
```

## LISTY W PYTHONIE

## KROTKI

```
empty = ()
```

```
# albo
```

```
empty = tuple()
```

```
print(empty) # ()
```

Sposób na puste słowniki/krotki

```
main.py
```

```
1 rozmiary = ("S", "M", "L", "XL")
2
3 print(len(rozmiary)) # 4
4 print(rozmiary[0]) # S
5 print(rozmiary[1:3]) # ('M', 'L')
6 print("XXL" in rozmiary) # False
7 print("S" in rozmiary) # True
8
```

```
4
S
('M', 'L')
False
True
```

A MOŻE TERAZ COŚ GRAFICZNEGO

---

**PYSIMPLEGUI AS SG**

### CO TO I DLACZEGO?

PySimpleGUI to prosta i intuicyjna biblioteka graficzna w języku Python, która umożliwia tworzenie interfejsów użytkownika (UI) w sposób szybki i łatwy. Jest to doskonałe narzędzie do tworzenia aplikacji desktopowych, które wymagają interakcji z użytkownikiem.

# CO TO I DLACZEGO?

- **Prostota:** PySimpleGUI został zaprojektowany tak, aby był prosty do zrozumienia i używania. Dzięki temu, nawet osoby, które nie są specjalistami od tworzenia interfejsów graficznych, mogą tworzyć aplikacje z GUI.
- **Wieloplatformowy:** Aplikacje PySimpleGUI mogą działać na różnych platformach, w tym Windows, macOS i Linux.
- **Szybkość tworzenia:** Dzięki PySimpleGUI można szybko stworzyć interfejs użytkownika i przetestować go w działaniu.
- **Wsparcie dla wielu narzędzi:** PySimpleGUI obsługuje różne narzędzia do tworzenia aplikacji desktopowych, takie jak Tkinter, Qt, WxPython, GTK i wiele innych.
- **Społeczność:** PySimpleGUI ma aktywną społeczność użytkowników i oferuje wiele przykładów i wsparcia online.

## INSTALACJA

**Raczej nie będzie tego bezpośrednio w naszym programie:**

- **pip install PySimpleGUI (w teorii, czasem musi być pip3)**

## START - POCZĄTEK PRACY Z APLIKACJĄ

**1. Importowanie biblioteki** - import PySimpleGUI as sg

**2. Zdefiniowanie layout** - np:

```
layout = [  
    [sg.Text("Witaj w mojej aplikacji!")],  
    [sg.Button("Kliknij mnie")],
```

```
]
```

#Definiujemy jak ma wyglądać okno - tu jest etykieta oraz przycisk

## START - POCZĄTEK PRACY Z APLIKACJĄ

### Tworzenie okna

Używamy sg.Window:

```
window = sg.Window("Moje Okno", layout)
```

Moje okno to tytuł, później odniesienie do stworzonego widoku

# START - POCZĄTEK PRACY Z APLIKACJĄ

## „Działanie okna”

Aby okno było interaktywne, czyli działało i obsługiwało zdarzenia, to trzeba zrobić pętlę:

while True:

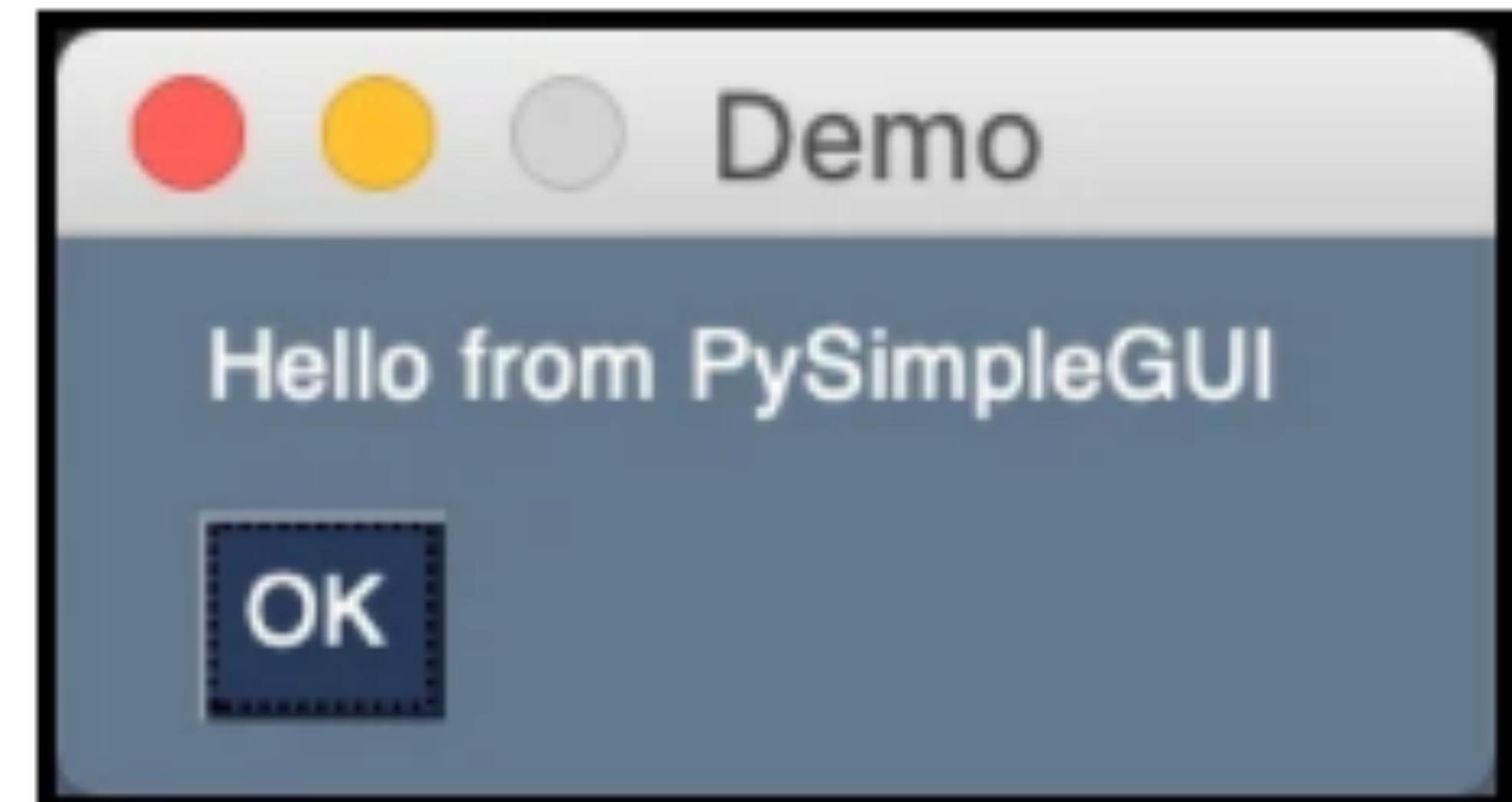
```
event, values = window.read()
```

```
if event == sg.WINDOW_CLOSED:
```

```
    break
```

```
if event == "Kliknij mnie":
```

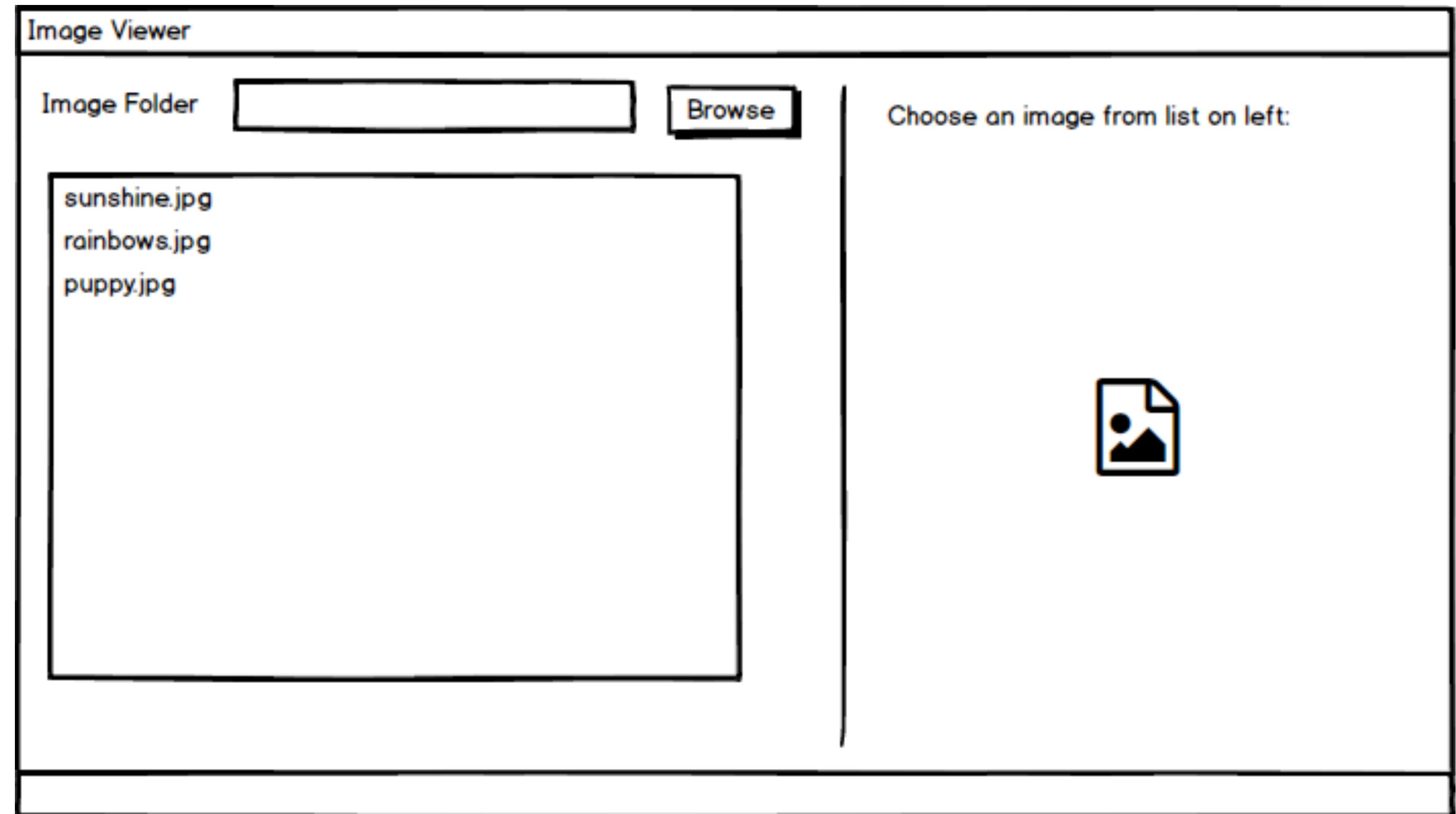
```
    sg.popup("Przycisk został kliknięty!")
```



## START - POCZĄTEK PRACY Z APLIKACJĄ

**„Zamknięcie okna”**

Zwykłe `window.close()`



# PIERWSZA BIBLIOTEKA GRAFICZNA

Image Viewer

Image Folder  [Browse](#)

Choose an image from list on left:

- sunshine.jpg
- rainbows.jpg
- puppy.jpg



KONIEC